Late Code Chunking: A Code Chunking Strategy for Repository-Level Code Completion

Anonymous ACL submission

Abstract

This paper introduces Late Code Chunking (LC^2), a chunking strategy designed to improve the semantic understanding of code segments for Large Language Models (LLMs). Repository-level code completion requires predicting the subsequent code for unfinished code by leveraging cross-file context spread across a repository. However, when retrieved fragments have missing semantics—the loss of structural or behavioral information during chunking—LLMs struggle to interpret the target code. To address this, LC^2 bifurcates chunks into a "Code Retrieval Context" optimized for similarity-based search and a "Code Comprehension Context" enriched via context expanding and augmentation. This dual-context design reduces information loss due to chunking and enhances the ability of LLMs to utilize retrieved code. Additionally, we introduce an Asymmetric Query-Chunk Sizing strategy to further enhance retrieval quality by minimizing query noise. Our experiments demonstrate that LC^2 provides robust performance enhancements, achieving a statistically significant 19.7% improvement in Exact Match accuracy on the CrossCodeEval benchmark compared to the best existing chunking method.

1 Introduction

Despite significant advancements in Large Language Models (LLMs) in code intelligence tasks, such as code generation and completion (Guo et al., 2024; Lu et al., 2021), **repository-level code completion** remains a challenging area. Traditional code completion leverages solely **in-file context** from the partially written code (Ciniselli et al., 2021). In contrast, real-world development frequently requires external knowledge spread across other files within the same repository—referred to as **cross-file context** (Liu et al., 2025). However, due to the limited context length of LLMs, it is infeasible to provide entire repositories as input. Instead, retrieval mechanisms must supply only the most relevant code fragments to the model. Benchmarks for repository-level code completion (Ding et al., 2023) are designed to evaluate this capability. Different Retrieval-Augmented Code Generation (RACG) (Su et al., 2024) approaches explore this task: RepoCoder (Zhang et al., 2023) iteratively refines a prediction by leveraging the prediction as a new query, RepoFormer (Wu et al., 2024) emphasizes filtering irrelevant cross-file content. GraphCoder (Liu et al., 2024b) introduces graph-based retrieval, DRACO (Cheng et al., 2024) constructs a background knowledge which consists of definition information called an unfinished code, and RLCoder (Wang et al., 2024) improves retrieval via reinforcement learning.

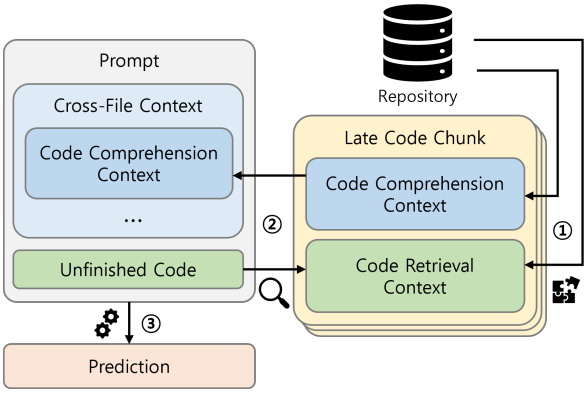

Figure 1: Pipeline of Late Code Chunking (LC^2) for repository-level code completion.

While these methods aim to find the most relevant cross-file context, **retrieval success does not guarantee generative success**. DRACO shows reduced performance when only short function signature information is leveraged compared to using the entire function content. A primary cause of failure is **semantic loss introduced during chunking**—the process of splitting source code into separately retrievable fragments (Günther et al., 2025). Missing structural or behavioral semantics leads

to misunderstandings of the retrieved code during generation. To overcome this inherent limitation, we propose **Late Code Chunking (LC^2)**, a **retrieval-aware** chunking strategy designed to enhance cross-file semantics. Figure 1 introduces a **dual-context** representation: A **Code Retrieval Context** contains a representative context for relevant matching to unfinished code, and a **Code Comprehension Context** contains detailed code implementation by expanding and augmenting the retrieved code to restore missing semantics. This dual-context approach mitigates code completion failures caused by information deficits. Additionally, LC^2 applies an **Asymmetric Query–Chunk Sizing** strategy, deliberately reducing the query length relative to the Code Retrieval Context size to maximize the opportunity for key-term matching (Tang et al., 2024).

Extensive experiments across six LLMs and four retrieval models demonstrate the effectiveness of LC^2. It showed **statistically significant improvements** on both RepoEval (Zhang et al., 2023) and CrossCodeEval (Ding et al., 2023) benchmarks, confirming its generalizability.

Our contributions are summarized as follows:

- We propose **LC^2**, a retrieval-aware dual-context chunking strategy tailored for repository-level code completion.
- We introduce **Asymmetric Query–Chunk Sizing** strategy to enhance retrieval performance and validate the effectiveness through extensive experiments.
- We validate LC^2 extensively across diverse LLMs and retrieval models, demonstrating consistent performance improvements.

The source code and more detailed experimental results of LC^2 are available at https://github.com/<*anonymous repository*>.

2 Methodology

LC^2 constructs two complementary representations for each chunk. We first generate a **Code Retrieval Context** designed for retrieval effectiveness, and then expand and augment it into a **Code Comprehension Context** that restores missing semantics.

2.1 Code Retrieval Context

Line-based chunking can capture natural structural boundaries in code. However, the uneven distribution of tokens per line introduces two significant problems: (1) when chunks become too short, leading to insufficient information for LLMs, or (2) when chunks exceed the allowed input size, forcing truncation and causing semantic loss (Liu et al., 2024a). To address these issues, LC^2 adopts **token-based chunking**, ensuring uniform distribution of semantic content across chunks.

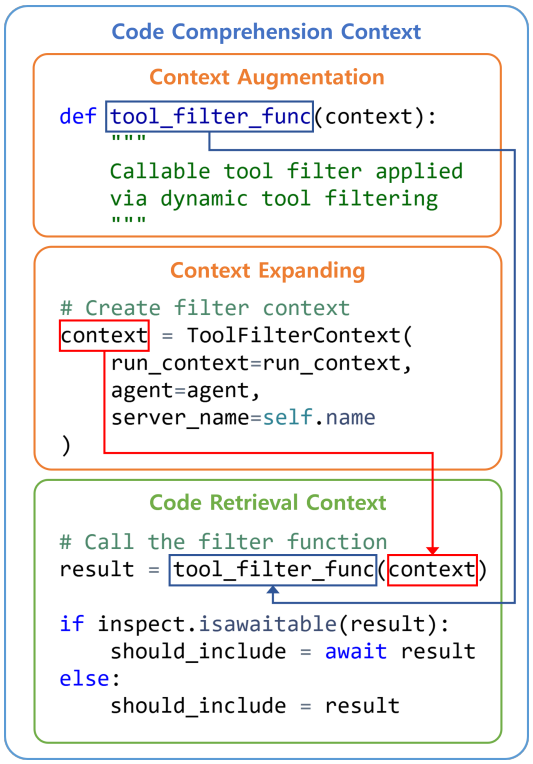

Figure 2: Example of Late Code Chunking.

Furthermore, in code completion, the most informative tokens typically appear near the end of the unfinished code. Thus, LC^2 employs an **Asymmetric Query–Chunk Sizing**, a short query but a larger retrieval chunk to maximize key-term matching opportunities while minimizing query noise, which significantly improves retrieval quality.

2.2 Context Expanding

The constitution of a given chunk fundamentally informs the preceding contextual information (Izadi et al., 2022); consequently, a high correlation exists between the content that immediately precedes a segment and the segment itself. As shown in Figure 2, a variable such as `context` appears without its preceding initialization, resulting in semantics missing. LC^2 addresses this by expanding the chunk to include preceding content that connects directly to the meaning of the retrieved code. This expansion partially restores lost semantics.

	RepoEval				CrossCodeEval					
	Line		API		Python		Java		C#	
Methods	**EM**	**ES**	**EM**	**ES**	**EM**	**ES**	**EM**	**ES**	**EM**	**ES**
In-File	31.13	64.58	30.12	64.89	5.82	59.31	6.87	59.59	3.34	58.15
Fixed-Window	43.19	71.94	41.81	72.79	18.99	67.45	17.67	64.77	16.18	65.33
Split-Aggregate	44.62	72.60	43.75	72.65	21.40	68.54	20.01	65.44	18.95	66.77
Function-Level	41.31	70.54	39.38	70.62	21.96	69.02	17.72	65.02	13.24	65.27
Fixed-Token	44.88	72.36	44.62	74.39	22.01	69.05	23.38	66.10	19.98	66.95
LC^2	**45.12**	**72.87**	**45.00**	**74.48**	**26.35**	**71.92**	**24.22**	**66.79**	**21.72**	**68.19**

Table 1: Performance of chunking methods on DeepSeekCoder-1.3B (Inference) and UniXcoder (Retrieval Model).

2.3 Context Augmentation

When a chunk contains a function call without its definition, LLMs struggle to reason about code behavior. LC^2 addresses this by analyzing the presence of function calls within the chunk, retrieving the corresponding function definition from the repository, and appending it to the Code Comprehension Context. To accommodate the limited context size of LLMs, inspired by DRACO (Cheng et al., 2024), only the function signature with a docstring is incorporated, enabling LLMs to utilize the critical API-level semantics. For instance, in Figure 2, the function `tool_filter_func` is referenced but not locally defined; LC^2 supplements its signature and docstring so that the LLM can leverage it as background knowledge.

3 Experimental Setup

3.1 Datasets

We evaluate LC^2 on two established benchmarks for repository-level code completion:

- **RepoEval:** A Python based benchmark. We utilized both the line and the API invocation datasets for the experiments.
- **CrossCodeEval:** A multilingual benchmark. We conducted experiments using datasets from three languages (Python, Java, and C#).

3.2 Evaluation Metrics

We adopt two widely used metrics:

- **Exact Match (EM):** A binary metric that assigns a score of 1 if the prediction is identical to the ground truth, and 0 otherwise.
- **Edit Similarity (ES):** A similarity metric derived from Levenshtein distance (Lcvenshtcin, 1966) for more fine-grained evaluation.

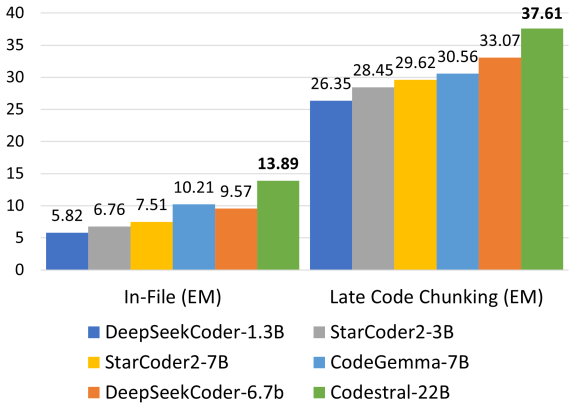

Figure 3: Performance comparison across LLMs on CrossCodeEval (Python) with EM metric.

3.3 Retrieval Models

We employ **BM25** (Robertson et al., 2009) as a representative sparse retrieval model. For dense retrieval, we use **UniXcoder** (Guo et al., 2022), and **CodeRank** models (Suresh et al., 2025) specialized in generating code semantic embeddings.

3.4 Chunking Strategies

We compare LC^2 against five baseline chunking approaches: **In-File** relies solely on the unfinished code within the current file. **Fixed-Window** uses line-based chunks (10 lines, following CrossCodeEval settings). **Split-Aggregate**, employed in the RLCoder framework, involves fine-grained line chunks merging when semantic fragmentation is detected. **Function-Level** extracts functions from a repository and utilizes each function as a chunk. **Fixed Token** structures chunks based on a uniform length (512 tokens).

LC^2 constructs a 512-token **Code Retrieval Context**, expands it by up to 512 additional tokens, and augments up to 3 functions with docstrings. The query consists of the last 64 tokens of the unfinished code under the asymmetric sizing strategy.

Retrieval Models	Params	EM	ES
BM25	-	26.20	72.29
UniXcoder	124M	26.35	71.92
CodeRankEmbed	137M	27.02	72.05
CodeRankLLM	7B	**29.47**	**72.95**

Table 2: Retrieval model performance comparison.

We found these parameters through experiments.

4 Experimental Results

4.1 Performance Comparison with Different Chunking Strategies

As presented in Table 1, Split-Aggregate, an enhanced strategy of Fixed-Window, achieves improved performance. Fixed-Token further improves performance through uniform semantic distribution. However, Function-Level shows unstable performance across different datasets and languages, mainly because many retrieved functions are either too short or lack context in isolation.

In contrast, LC^2 consistently achieves the best performance in all settings. Notably, LC^2 improves EM by **19.7%** on CrossCodeEval (Python) compared to Fixed-Token, the strongest baseline. These results confirm that recovering cross-file semantics is crucial for repository-level code completion.

4.2 Evaluation Across LLMs and Retrieval Models

To assess generalizability, we evaluate LC^2 with six LLMs and four retrieval models. Figure 3 illustrates the EM performance gains on CrossCodeEval (Python) with UniXcoder when LC^2 is applied compared to the In-File baseline. Codestral (Mistral, 2024) achieves the highest overall scores (EM: 13.89 → 37.61). Among the 7B parameter models—StarCoder (Lozhkov et al., 2024), CodeGemma (Team et al., 2024), and DeepSeekCoder (Guo et al., 2024), DeepSeekCoder exhibits the most significant gain (EM improvement: +23.50), this demonstrates that LC^2 enhances models regardless of parameter scale or pretraining architecture.

Regarding retrieval models, as shown in Table 2, state-of-the-art CodeRank models outperform UniXcoder in dense retrieval performance evaluations. A notable observation is that BM25, still widely used for lightweight sparse lexical retrieval, demonstrates performance nearly comparable to dense retrieval approaches. This result empirically validates the importance of exact lexical matching (Gao et al., 2021) in code completion and demonstrates the effectiveness of our Asymmetric Query-Chunk strategy.

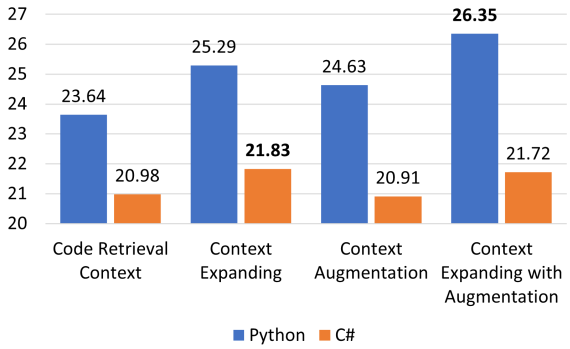

Figure 4: Analyze the contribution of each LC^2 component on CrossCodeEval (Python) with EM metric.

4.3 Ablation Study

As shown in Figure 4, the **Code Retrieval Context** with the Asymmetric Query-Chunk Sizing strategy already surpasses all baselines, listed in Table 1. Although the maximum embedding length of UniXcoder is constrained to 512 tokens, **Context Expanding** enables the generation of chunks that exceed this limit, further refining the semantics of the retrieved context. Applying **Context Augmentation**, which appends function signatures with docstrings for enabling better reasoning about external dependencies, yields additional performance improvements. However, in particular experimental environments such as C#, we observed performance degradation when appended functions were non-informative. The result aligns with the trends observed in Function-Level baselines from Table 1, confirming that irrelevant augmentation introduces semantic noise, even when only function signatures are added. Enhancing selective augmentation strategies remains a topic for future work.

5 Conclusion

We presented LC^2, a chunking strategy tailored for repository-level code completion that effectively mitigates semantic loss induced by conventional chunking. Our study opens exciting avenues for future research in this field.

Future work includes developing selective augmentation techniques to reduce the injection of irrelevant context and extending LC^2 to broader code intelligence tasks, such as code generation for natural language queries and automated bug fixing.

Limitations

When applying the *context augmentation* technique of LC^2 to enrich information of function calls, performance exhibited both improvements and declines depending on the contextual characteristics of the repository. Although the Ablation Study revealed the root causes, further research is necessary to identify performance-degrading factors through empirical validation of case studies and to develop a methodology for selectively incorporating only those functions that contribute to performance improvement.

References

Wei Cheng, Yuhan Wu, and Wei Hu. 2024. Dataflow-guided retrieval augmentation for repository-level code completion. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7957–7977, Bangkok, Thailand.

Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. 2021. An empirical study on the usage of bert models for code completion. *Preprint*, arXiv:2103.07115.

Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. 2023. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. *Preprint*, arXiv:2310.11248.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. COIL: Revisit exact lexical match in information retrieval with contextualized inverted list. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3030–3042, Online. Association for Computational Linguistics.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, Dublin, Ireland.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Michael Günther, Isabelle Mohr, Daniel James Williams, Bo Wang, and Han Xiao. 2025. Late chunking: Contextual chunk embeddings using long-context embedding models. *Preprint*, arXiv:2409.04701.

Maliheh Izadi, Roberta Gismondi, and Georgios Gousios. 2022. Codefill: multi-token code completion by jointly learning from structure and naming sequences. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 401–412, New York, NY, USA. Association for Computing Machinery.

VI Lcvenshtcin. 1966. Binary coors capable or 'correcting deletions, insertions, and reversals. In *Soviet physics-doklady*, volume 10.

Jiaheng Liu, Ken Deng, Congnan Liu, Jian Yang, Shukai Liu, He Zhu, Peng Zhao, Linzheng Chai, Yanan Wu, JinKe JinKe, Ge Zhang, Zekun Moore Wang, Guoan Zhang, Yingshui Tan, Bangyu Xiang, Zhaoxiang Zhang, Wenbo Su, and Bo Zheng. 2025. M2RC-EVAL: Massively multilingual repository-level code completion evaluation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15661–15684, Vienna, Austria.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Wei Liu, Ailun Yu, Daoguang Zan, Bo Shen, Wei Zhang, Haiyan Zhao, Zhi Jin, and Qianxiang Wang. 2024b. Graphcoder: Enhancing repository-level code completion via code context graph-based retrieval and language model. *Preprint*, arXiv:2406.07003.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, and 47 others. 2024. Starcoder 2 and the stack v2: The next generation. *Preprint*, arXiv:2402.19173.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou, Nan Duan, Neel Sundaresan, and 3 others. 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

AI Mistral. 2024. Codestral, 2024. *URL: https://mistral.ai/news/codestral*.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Hongjin Su, Shuyang Jiang, Yuhang Lai, Haoyuan Wu, Boao Shi, Che Liu, Qian Liu, and Tao Yu. 2024. EvoR: Evolving retrieval for code generation. In

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 2538–2554, Miami, Florida, USA.

Tarun Suresh, Revanth Gangi Reddy, Yifei Xu, Zach Nussbaum, Andriy Mulyar, Brandon Duderstadt, and Heng Ji. 2025. CoRNStack: High-quality contrastive data for better code retrieval and reranking. In *The Thirteenth International Conference on Learning Representations*.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Forty-first International Conference on Machine Learning*.

CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A. Choquette-Choo, Jingyue Shen, Joe Kelley, Kshitij Bansal, Luke Vilnis, Mateo Wirth, Paul Michel, Peter Choy, Pratik Joshi, Ravin Kumar, Sarmad Hashmi, Shubham Agrawal, and 8 others. 2024. Codegemma: Open code models based on gemma. *Preprint*, arXiv:2406.11409.

Yanlin Wang, Yanli Wang, Daya Guo, Jiachi Chen, Ruikai Zhang, Yuchi Ma, and Zibin Zheng. 2024. Rlcoder: Reinforcement learning for repository-level code completion. *Preprint*, arXiv:2407.19487.

Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. 2024. Repoformer: Selective retrieval for repository-level code completion. *Preprint*, arXiv:2403.10059.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2471–2484, Singapore.